

AD-A055 597

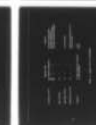
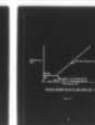
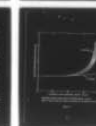
DEFENSE SYSTEMS MANAGEMENT SCHOOL FORT BELVOIR VA
ESTIMATING COMPUTER SOFTWARE DEVELOPMENT COSTS.(U)
1975 A W ANDRES

F/G 5/1

UNCLASSIFIED

NL

1 OF 1
ADA
055597



END
DATE
FILMED
8 -78
DDC

FOR FURTHER TRAN *11.11*

1

AD-A055597

DEFENSE SYSTEMS MANAGEMENT SCHOOL



PROGRAM MANAGEMENT COURSE INDIVIDUAL STUDY PROGRAM

ESTIMATING COMPUTER SOFTWARE DEVELOPMENT
COSTS

STUDY REPORT
PMC 75-1

ALBERT W. ANDRES
GS-13 DNC

DDC
RECEIVED
JUN 23 1978
D

FORT BELVOIR, VIRGINIA 22060

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

ANDRES

DISCLAIMER NOTICE

**THIS DOCUMENT IS BEST QUALITY
PRACTICABLE. THE COPY FURNISHED
TO DDC CONTAINED A SIGNIFICANT
NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.**

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ESTIMATING COMPUTER SOFTWARE DEVELOPMENT COSTS		5. TYPE OF REPORT & PERIOD COVERED Study Project Report 75-1
7. AUTHOR(s) ALBERT W. ANDRES		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS DEFENSE SYSTEMS MANAGEMENT COLLEGE FT. BELVOIR, VA 22060		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS DEFENSE SYSTEMS MANAGMENT COLLEGE, FT. BELVOIR, VA		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) UNLIMITED		12. REPORT DATE 1975-1
		13. NUMBER OF PAGES 67
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
16. DISTRIBUTION STATEMENT (of this Report) UNLIMITED		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
<div style="border: 1px solid black; padding: 5px; text-align: center;"> DISTRIBUTION STATEMENT A Approved for public release; Distribution Unlimited </div>		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES SEE ATTACHED SHEET		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) SEE ATTACHED SHEET		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		

DEFENSE SYSTEMS MANAGEMENT SCHOOL

STUDY TITLE: ESTIMATING COMPUTER SOFTWARE DEVELOPMENT COSTS

STUDY PROJECT GOALS:

To identify, define, and evaluate techniques for estimating computer software development costs.

Demonstrate how these techniques apply to estimating in the DOD Program Office.

STUDY REPORT ABSTRACT

The purpose of this study project was to increase the reader's knowledge of computer software development cost estimating. The reasons for the high cost of computer software and poor computer software estimates are discussed. Existing estimating techniques and rules of thumb are presented and their applicability to cost estimating minicomputer software development in the Program Office is considered.

The conclusion is that there is no technique that will give an accurate estimate for all situations. A large number of factors that affect the cost of computer software development have been identified. Therefore, it is mandatory that the estimator have the best possible handle on the most important of these factors.

The most significant recommendation that this report makes is that the Program Office establish data bases from which estimates can be made by parametric methods.

This report has implications for anyone involved in the procurement of computer software.

SUBMISSION TO	
DTIC	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFICATION.....	
BY.....	
DISTRIBUTION/AVAILABILITY CODES	
Dist.	AVAIL. and/or SPECIAL
A	228

NAME, RANK, SERVICE

Albert W. Andres, GS-13, DDC

CLASS

PMC 75-1

DATE

May 1975

ESTIMATING COMPUTER SOFTWARE DEVELOPMENT
COSTS

STUDY REPORT

Presented to the Faculty
of the
Defense Systems Management School
in Partial Fulfillment of the
Program Management Course
Class 75-1

by
Albert W. Andres
GS-13 DNC

This study represents the views, conclusions and recommendations
of the author and does not necessarily reflect the official opinion
of the Defense Systems Management School nor the Department of Defense.

EXECUTIVE SUMMARY

The purpose of this study project was to increase the reader's knowledge of computer software development cost estimating. Its specific goals were: 1) to identify, define, and evaluate techniques for estimating computer software development costs, 2) to demonstrate how these techniques apply to estimating in the DOD Program Office. Emphasis was placed on minicomputer software.

This study area is important because DOD computer software involves billions of dollars each year and a large percentage of cost growth is because of poor cost estimating.

For background, the reasons for the high cost of computer software and poor computer software estimating were investigated. Existing estimating techniques and rules of thumb were considered. Although certain trends can be established, there is no technique that will give an accurate estimate for all situations. There are many factors that affect the cost of computer software development. Unless the estimator has a very good handle on most of these factors as they relate to the proposed programming effort his estimate will be a poor one.

Only a ball park estimate can be made without a good historical data base to work from. The primary recommendation of this study is that the Program Office make an intensive effort to establish a data base which is broken down into as many cost contributing factors as possible.

This report should be useful to anyone involved in the procurement of computer software.

ACKNOWLEDGEMENTS

I wish to express my appreciation to Ed Rappe of the Defense System Management School and Ray LeSage of the Naval Electronic Systems Command for their constructive criticism.

CONTENTS

Executive Summary	ii
Acknowledgements	iv
I. Introduction	1
II. Review of Present Situation	9
III. Data Collection Method	11
IV. Reasons for High Software Costs	12
V. Reasons for Poor Cost Estimates for Computer Software Development	16
VI. The Cost Estimate	18
VII. Conclusions and Recommendations	37
Bibliography	39
Appendix - Glossary of Digital Computer Terms	A-1

LIST OF ILLUSTRATIONS

Figure		Page
1.	Comparison of Programmable Calculator, Minicomputer, and Midicomputer Characteristics	2
2.	Computer Programming Project Cycle	5
3.	Hardware Strains Cause Major Software Impact	13
4.	System Development: Reliability of Estimates	19
5.	Computer Programming Productivity in Instructions per Manhour	21
6.	Manmonths Verses Program Size for Eleven Large-Scale Programs	22
7.	Computer Software Documentation Productivity Information	25
8.	Relation of Documentation Cost to Total Project Cost	26
9.	Computer Time for Programming	27
10.	Software Development Cost Breakdown, Tasks as a Percent of Total Effort	29
11.	Krauss's Values for Input Variables	31

I. INTRODUCTION

Purpose

The purpose of this study project was to increase the reader's knowledge of computer software development cost estimating. Its specific goals were: 1) to identify, define, and evaluate techniques for estimating computer software development costs, 2) to demonstrate how these techniques apply to estimating in the DOD Program Office. Emphasis was placed on minicomputer software.

Scope and Limitations

Because of the increasing use of minicomputers in the Department of Defense, this paper will address cost estimating of minicomputer software development in particular.

Important Terms

Minicomputer

What is a minicomputer? Figure 1 shows a comparison of programmable calculator, midicomputer, and minicomputer nominal characteristics.

An expansion of the minicomputer characteristics is presented below.

Processor

Usually single address, 8 to 18 bit word size (usually 12 or 16).

Memory

Usually core, 600 nanosecond to 1 microsecond cycle times, 1024 to 32,768 words (some memories can be expanded to 131,072 words).

Characteristics	Programmable Calculator	Minicomputer	Midi or Small Computer
Maximum memory word size	100	32,000	256,000
Maximum number of bits per word	64	18	24
Higher-level language	Hardware: BASIC	Software: FORTRAN BASIC ALGOL RPG	Software: FORTRAN BASIC ALGOL RPG COBOL
Function	Dedicated	General-Purpose	General-purpose
Display I/O, and recording devices	Built-in	External peripherals	External peripherals
Speed	Slow	Fast	Fast
Programming	Manually from integral keyboard	Assembly or higher-level language	Higher-level language
Required user knowledge of machine-level operation	None	Extensive	Limited
Applications	Dedicated program solving Limited data acquisition	Limited time sharing Problem solving Data Acquisition Process Control Peripheral Control	Simultaneous limited time sharing and batch processing Multiprogramming Data acquisition Process control Extensive problem solving
Cost	Very low	Low	High

Figure 1. Comparison of programmable calculator, minicomputer, and midicomputer characteristics (1:18)¹

¹This notation will be used throughout the report for major references. The first number is the source listed in the bibliography. The second number is the page in the reference.

Input/Output

Flexible

Software

Software for minicomputer systems can be divided into several classes: (1:10)

1. Program development software needed by the user to develop his programs for particular applications. Consists of editors, assemblers, debugging and utility routines, and one or several compilers such as BASIC or FORTRAN.
2. Input/Output software routines for the system hardware and peripherals. These packages are generally defined by the characteristics of the respective hardware.
3. Operating system software, also called the executive or system monitor, controls the operation of the minicomputer system.
4. Applications software, which is related to the task that the system is to perform and which is therefore unique to the particular system.

Peripherals

Card readers, paper tape readers, line printers, interactive terminals (keyboard/printer or keyboard/display), magnetic tape units, disk and drum memories.

Physical Size

The typical mini weighs less than 200 lbs., occupies less than 4 cubic feet, and is very undisturbed by reasonable power or heat variations. (2:3-4)

Cost

Commercial minicomputers cost less than \$20,000 and often much less depending on the capability required. The price of the Navy standard minicomputer (AN/UYK-20) is in the \$30,000 to \$40,000 range depending on the needed capability.

Computer Software Development Life Cycle

The steps making up the computer programming process, or project cycle, are assumed to be as defined by Nelson (3). See Figure 2. The six steps are:

Preliminary Planning and Cost Evaluation

This activity consists of the economic feasibility study for the proposed program. Based on a statement of the user's requirements, an estimate is made of the manpower, elapsed time, or other resources required for the project. Using these estimates, a summary project plan and a cost verses benefits comparison are prepared.

Information Processing System Analysis and Design

The process of determining the detailed requirements for improved information processing and planning of a system, plus a set of computer programs capable of fulfilling them, is divided into two parts--System Analysis and System Design. The first part, the analysis (sometimes called problem formulation), consists of investigating the particular information processing problem that may be solved by new or improved automatic data processing methods; the second design consists of attempts to devise a satisfactory solution

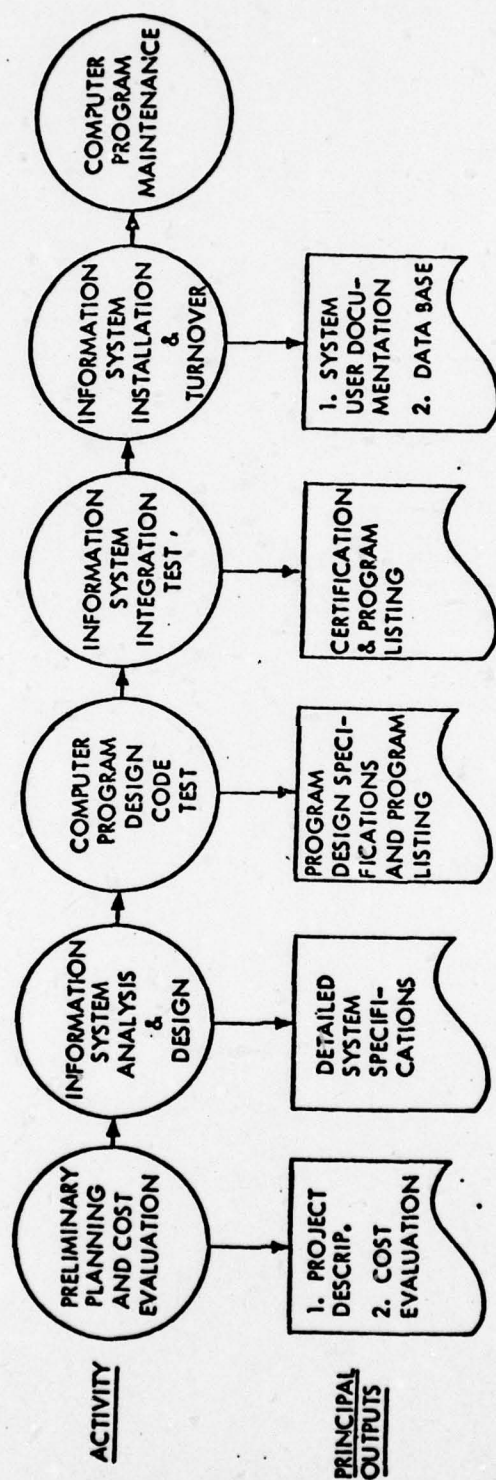


Figure 2. Computer Programming Project Cycle (3:6)

to the data processing requirements involved. In the broadest sense, the problem and its solution may involve the design of a far-flung network including communications displays, data equipment for sensors or missiles, computer operating procedures, and computer programs. In its narrowest sense, Analysis and Design work as part of computer programming may only include the design of a change to a computer program in an existing system.

Generally, the mission of the analyzing and synthesizing process is to devise the most effective and efficient organization of system components including computer program functions and elements possible within the constraints of available manpower, funds, and time, to perform the required information processing functions. Ideally, this selection of a solution should be made on the basis of cost/benefit comparison of feasible alternatives.

Computer Program Design, Code, and Test

This activity covers all work necessary to produce the computer program end product in accordance with the detailed specification of requirements for the computer program including design, code, test (debug) and documentation work for the entire program as well subprograms (runs, segments, individual programs).

Information Processing System Integration Test

This activity covers all work necessary to test the performance of the computer program within the total system at the operational facility under realistic ("live") operating conditions.

Information Processing System Installation and Turnover

The purpose of the turnover step is to help the user demonstrate, at his own operational site, that the computer program system will operate as specified, and to support the user with documentation, advice, and guidance, and troubleshooting during the initial period of system operation.

Computer Program Maintenance

Computer program maintenance is the process of improving, changing, and correcting computer programs in an information system that is currently operational.

Program maintenance, including both revisions and error corrections, is needed throughout the life of the information system. Revisions are needed because operational requirements are continually changing during both the development and operation of the system. Although operational needs are projected during requirements analysis, in most cases they can be neither totally defined nor totally implemented in the imposed time schedules. Also, corrections must usually be made to the computer programs because errors and operational deficiencies not detected in the routine testing of the programs are usually discovered when the system becomes operational.

Since the need for improvement and support activities for the information system tends to be amorphous, system maintenance is often funded at a level the user can afford or is willing to spend rather than the level precisely required. Much of the work of program maintenance personnel must be devoted to the resolution of emergencies

and to modifications required by hard-to-predict environmental changes.

Organization of the Report

In the analysis of minicomputer software development cost estimating, reasons for high software costs and poor estimates will be considered. Existing methods of computer software estimating will be evaluated to determine applicability to estimating in the DOD Program Office.

II. REVIEW OF PRESENT SITUATION

The purpose of this section is to document the importance and the present status of computer software development cost estimating.

Sources of Cost Growth

In this age of highly complex equipment and continuing pressure to incorporate technology pushing the "state of the art" overruns in both cost and development time are more the rule than the exception. (4:4) The experiences of more than 30 major programs over an extended period of time give some indication of the problems of cost growth most likely to beset new programs. These program histories show that the factors contributing to cost growth and their approximate impact were:

1. Changes in Cost Estimates-refinements of the base program estimate-accounted for 40 percent of the total cost growth.
2. Engineering Changes-alterations in physical or functional characteristics-20 percent.
3. Schedule Changes-changes in delivery schedules or program milestones-15 percent.
- 4.. Economic Changes-escalation adjustments in contracts and other changes in the purchasing power of the dollar-10 percent.
5. Support Changes-changes in spare parts, training, testing, and other support requirements-7 percent.

A variety of other items made up the balance of some 8 percent of the total cost growth in these programs. (5:39)

High Cost of Computer Software

Estimates of current Air Force annual expenditures on software are

between \$1 billion and \$1.5 billion, compared to \$300 to \$400 million per year on computer hardware. (6:4) The total DOD expenditure would therefore be quite substantial.

The concern of the Air Force about computer software costs for weapons systems is documented by their Project Ace progress reports. (7:69,93)

The combination of changes in cost estimates being a large percentage of the total DOD program cost growth and the high cost of computer software, make computer software cost estimating an inviting subject.

Minicomputer Trend

Minicomputers are becoming serious competition for the large computers. Several minis put together can outperform a large computer at a fraction of the hardware cost. (8:11) The number of minicomputers available is rising rapidly; as prices continue to decline, an increasingly greater number of applications are discovered. Assuming a healthy economic climate the market should sustain 30-40% growth rate over the next few years. (2:3-1) DOD will be a contributor to this growth rate.

Estimating Difficulty

It is extremely difficult to estimate the time and effort necessary to produce a computer software package. This difficulty stems from a poor understanding of the production process, the number of disparate factors affecting program complexity, and the limited ability of human decision-makers to assimilate and effectively weigh these factors. Studies concerning software estimation have identified at least 90 factors that affect the overall cost of computer program development. (9:v)

III. DATA COLLECTION METHOD

To determine what computer software estimating techniques and rules of thumb are in use today, the following sources were used.

1. Defense Documentation Center computer search.
2. Review of current civilian literature.
3. Interviews with representatives of DOD Program Offices utilizing minicomputers as a subsystem.
4. Interviews with computer software consultants.
5. Questionnaire response/interview with contractors involved in programming minicomputers.
6. Interviews with minicomputer manufacturer's representatives.
7. Interviews with representatives of a DOD office that procures minicomputers and associated software.

Because of the relative newness of the minicomputer most of the data obtained was related to large programming efforts. However even the large program information is considered to be relevant to the study of the minicomputer problem in that trends can be shown by looking at the overall problem. This will be demonstrated in later sections of this paper.

IV. REASONS FOR HIGH SOFTWARE COSTS

As stated in Section II, computer software is expensive. The purpose of this section is to present some of the reasons for the higher than expected costs. Depending on the circumstances, cost may be reduced by attacking the cause.

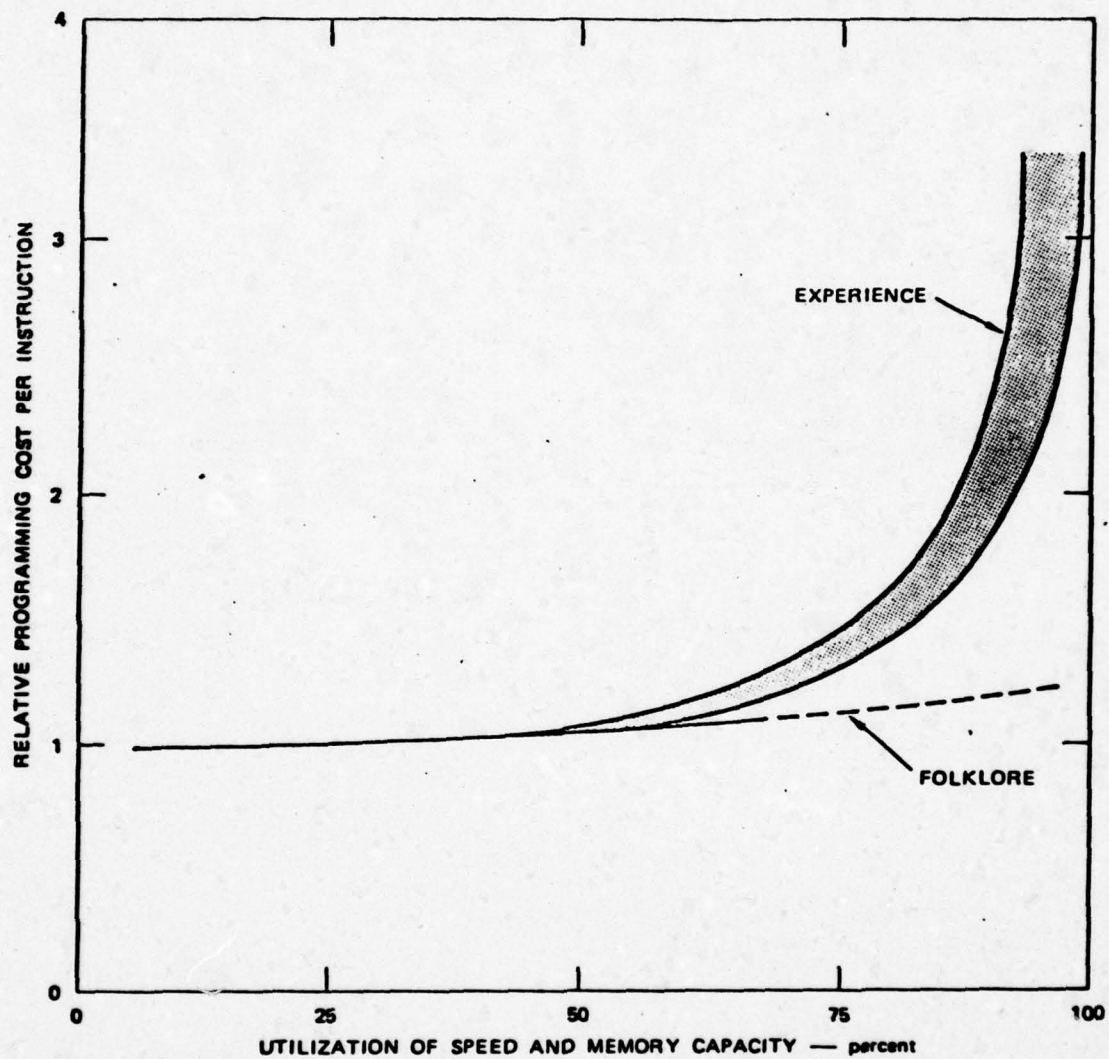
Poor Cost Estimates

An over optimistic estimate is probably the largest factor in the computer software surprise costs.

Memory Size Constraints

When a programmer gets into a situation where he has to be concerned with computer memory size the programming cost will increase. This is shown graphically in Figure 3. When the computer's capacity is pushed, machine language must be used instead of high-level languages such as FORTRAN; several elements of data must be packed into a single machine word, and many tricky programming shortcuts must be employed. This makes the program more difficult to check out and modify as well as more difficult to write.

Overall system cost is generally minimized by procuring computer hardware with at least 50 percent to 100 percent more capacity than is absolutely necessary. The more the ratio of software-to-hardware cost increases (as it will markedly during the 1970's and 1980's), the more excess computing capacity one should procure to minimize the total cost. It is far more risky to err by procuring a computer that is too small than one that is too large. This is especially important since the initial sizing of the data-processing job often tends to underestimate its magnitude. (10:49) Memory size constraints are particularly relevant to mini-computers because the memory is small.



HARDWARE STRAINS CAUSE MAJOR SOFTWARE IMPACT (10:47)
(Effect of hardware speed and memory size constraints on the relative cost of software)

Figure 3.

Change in Requirement

A fantastic increase in a weapons system capability is obtained when computer control/analysis is added. When the user sees this increased capability he may want more. Also, in many applications a hardware/software trade off exists. The hardware design is frozen relatively early in the development phase, presenting a tempting option to change software requirements in order to meet changing operational requirements or hardware deficiencies.

Hardware/Software Incompatibility

There is much waste in programming and computing, resulting from poor matching of software and hardware. Also, poorly designed primitive functions in hardware require repeated costly and error-inducing programming of basic computational functions. (11:1)

Indirect Costs

Software is often a critical component in large systems. Consequently, overruns in delivery time or serious flaws in quality can have hidden system costs that far exceed the software costs, however high they may be. (11:1)

Inadequate Functional Specification

Because of eagerness to get things moving the Government is susceptible to providing, and the contractor accepting, a functional specification that is not adequate. Accordingly, the contractor estimates low, a commitment is made and we are underway toward a potential overrun situation. The Programm Office and the contractor must have a clear and documented understanding of the user's requirement.

Production Demands

Progress in software technology has been very slow, but demands for software production are increasing in volume and complexity. Such demands have clearly outstripped the technology, with very costly results. Production of new software products suffers great overruns in cost and delivery time, and quality is often deficient, in correctness, modifiability, and transferability. The maintenance costs for old software products may be an order of magnitude larger than production cost, due to poor original design and production. (11:1)

Variations in Programmer Efficiency

The differences in computer programmer productivity can vary as much as 26:1 between individuals. (12:52) Higher than expected costs will result if estimates are made assuming that a highly efficient programmer will do the work but a very inefficient programmer actual does it.

V. REASONS FOR POOR COST ESTIMATES FOR COMPUTER SOFTWARE DEVELOPMENT

Almost everyone in a software development management position is aware that a large percentage of estimates are poor, but probably not many are aware of the reasons why. This section will present some of the reasons.

Keider (13:53) reports the following reasons for poor computer software development estimates.

1. No standards exist for estimating how long the project will take. That is, each project is treated as a new and novel system with some individual responsible for estimation. His estimate will be based upon his own understanding of the project and its tasks, and on how quickly he can accomplish the subtasks. Little use is made of a history file of similar projects and actual versus originally estimated times.
2. Estimation is not done by the probable project leader, but rather, by whoever happens to be available at estimating time.
3. The project is not adequately defined. The request for an estimate usually takes the form of "John, we're planning to redo the payroll system. What do you think it will require?" "Payroll" may mean a number of different things to different people. Does it involve labor distribution? Personnel information? leave accounting? salary, hourly and executive payroll? Any of the above can measurably impact the estimate of the project.
4. Short lead times are allowed for estimates, with corresponding inaccuracy as a result.
5. Knowledge of "tools" to perform the project more efficiently is lacking. Are there modules, or subroutines already available which can be used? Is there a test data generator available? What about system design or documentation aids?
6. Definition of the project is vague, misleading, or totally wrong.

Use of Nonrelevant Historical Data

The experience of many program managers is that the number of instructions is often grossly underestimated except when very similar programs can be used for comparison. (14:242) As will be shown later in this report, there are numerous variable factors that impact software productivity. These factors must be compared in detail between the programs on which the historical data is based and the program that is being estimated.

VI. THE COST ESTIMATE

The purpose of the section is to analyze existing computer software cost estimating data, equations, and rules of thumb to determine their applicability to DOD minicomputer software.

The computer software estimating effort can be assumed to consist of three steps.

1. Determining the scope (or number of instructions in the proposed program) and complexity from a functional or design specification.
2. Converting the number of instructions into quantity of resources required in programmer man months, computer time etc.
3. Converting the effort into dollars.

Step 1 - Determine the Scope

As a lead-in to this step the DOD Program Office should assure that the best possible functional specification exists, and it is even desirable to have a detailed design specification. The Computer Program Performance Specification and Computer Program Design Specification should be contracted for separate from the actual programming effort. The more that is known, the better the estimate should be. Figure 4 was prepared with the assumption that at each project stage, estimates are made for all of the successor stages. The preliminary analysis stage in Figure 4 can be equated to the preliminary planning and cost evaluation stage described in Section I. Assuming that the data in Figure 4 has some validity, it is apparent that the results of estimating in the early stages should be handled with caution.

PROJECT STAGES	Preliminary Analysis	Systems Analysis	Programming	Operating
Initial Approval	<u>+5%</u>	<u>+25-50%</u>	<u>-10+100%</u>	-
Preliminary Analysis	-	<u>+15-25%</u>	<u>+50_100%</u>	<u>-10+100%</u>
Systems Analysis	-	-	<u>+20_25%</u>	<u>+50_100%</u>
Programming	-	-	-	<u>+10%</u>

Figure 4. System Development: Reliability of Estimates for Successive Project Stages. (15:3-01-04)

Determining the size and complexity of a new program is considered by many to be black magic. To assure a reasonable degree of success in this task the following course of action should be taken.

1. Obtain the best possible estimator or estimating team.
2. The estimator should get a thorough understanding of the performance specification.
3. Identify existing programs which may be similar in function, size, and complexity to the program being estimated.
4. After a thorough comparison of the new and existing programs to insure that a good understanding of the similarities and differences exists, a scaling operation can take place. Because of the number of variables which impact the scope of the job, close attention must be given to the differences between the existing and new programs and compensate for those differences.

The key is to obtain programs that are as similar as possible to the new program.

Step 2 - Determine Resources Required

This step can be initiated after the basic scope of the proposed program has been determined. That is, the estimated number of instructions in the program and complexity have been established.

With the number of instructions as a starting point an estimate of resources required can be accomplished by parametric methods. Parametric estimates are determined by relationships between historical costs and physical and/or performance characteristics.

In an attempt to determine what minicomputer software programming rates can be expected Figure 5 was generated. A quick look at the data indicates large variations in the various rules of thumb. A closer look shows that certain trends exist.

Both the Aron and Nanus/Farr data show a large variation in programming productivity with variations in the size of the program. Figure 6 indicates an exponential rate of increase in man months of effort required verses the size of the program. Aron's data gives us one reason for this trend. The larger the program, the greater the number of programmers required, therefore more interactions are required. The greater the number of interactions, the poorer the efficiency. This theory is supported by Brooks. (16)

The interactions between programmers should be minimized on minicomputer programming efforts. Because of the small programs, typically only one to three programmers would be involved. Therefore, it is assumed that a nearly linear relationship exists between the programming effort

	Min	Max	Mean	Comments
Nanus/Farr (14:242)	2.38	5.95	4.16	small programs (less than 10,000 instructions)
			1.19	large programs
AN/UYK-7 Report (17:Bl)			1.46	
Wood (18:152)			.83	
CCIP-85 (6:12)			1.25	assumes large programs
Aron (16:49)			.74	many interactions
			2.48	some interactions
			4.96	very few interactions
Fleishman (19)	.49	43.15	4.89	scientific application, program sample-27
	.15	82.67	9.05	business application, program sample-79
	.06	43.15	3.63	machine oriented language,
	.63	82.67	11.76	procedure oriented language,
Bell Labs (16:49)			.255	complex, 52,000 word program, 83 programmers
			.313	complex, 51,000 word program, 60 programmers
			1.10	less complex, 38,000 word program, 9 programmers
			1.13	less complex, 25,000 word program, 13 programmers
Corbato (20:115)			.60	large program, high level language
Company A			.89	medium to low risk
			.71	high risk
Company B			.5	assembly language, highly technical/real time program, good programmers
Company C			2.25	21,000 instructions, FORTRAN-5,000 Assy-16,000
Company D			1.0	
Company E			1.27	no documentation, 7,500 instructions, assembly
Company F			.744	real time, executive program, assembly language
			.99	non real time
			1.49	modified program

Figure 5. Computer Programming Productivity in Instructions per Manhour.

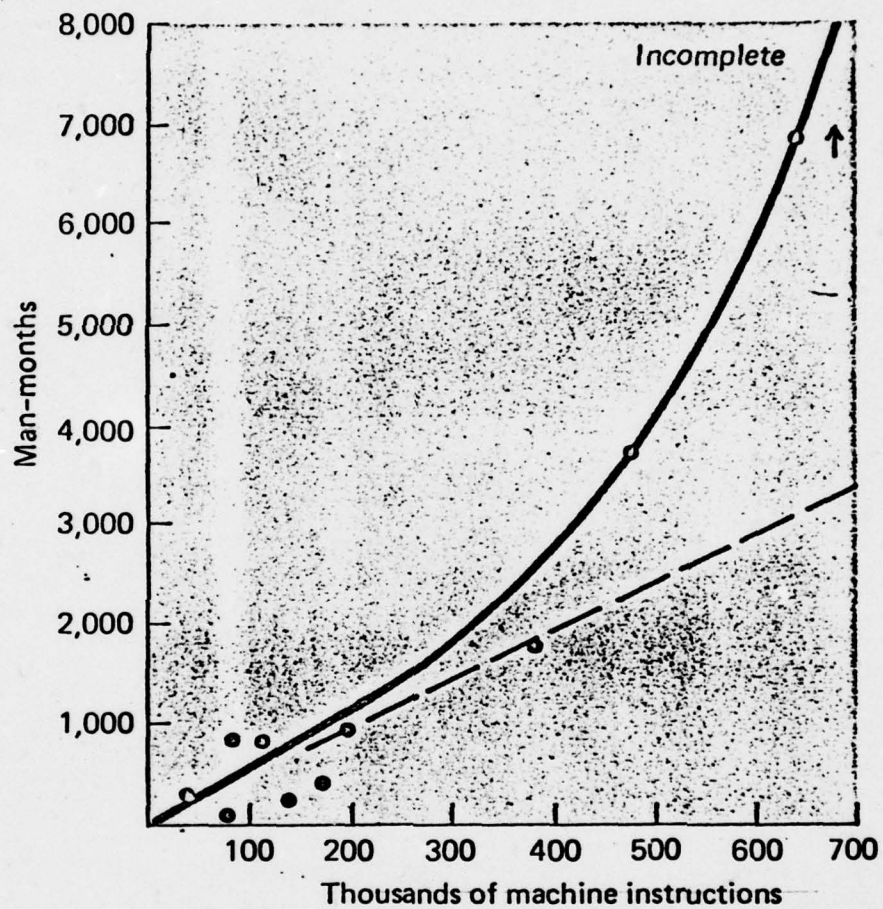


Figure 6. Manmonths Versus Program Size for Eleven Large-Scale Programs (14:242)

required and the size of the program for a minicomputer program.

The Fleishman data was included in Figure 5 to show the general large variations in programming efficiency. This data does show the expected trend in efficiency improvement in going from machine oriented language to procedure oriented language, and the decrease in efficiency in going from a business application to a scientific application.

The Figure 5 data labeled Company A through F was obtained from government contractors for minicomputer software development. This data was obtained informally rather than by audits. The approximate average of one instruction per man hour is lower than might be expected, however the data does include the effort required for documentation (except company E) and much of the programming was in assembly language.

The conclusions that can be made from Figure 5 are:

1. Certain productivity trends can be identified.
2. Productivity rates are dependent on many variables, therefore no one productivity rate can be used as a rule of thumb for accurate estimating.
3. Ball Park estimates can be made for minicomputer software development by using a one instruction per man hour productivity rate. This estimate must be refined of course, if such things as type of language to be used, complexity, and individual programmer efficiency are known.

Documentation

The effort required to document computer software is included in the total effort of Figure 5, but because the level of documentation desired may range from full MilSpec (such as in accordance with Weapons Specification WS-8506 and DOD Manual 4120.17-M) to back of the envelope notes, it

should be considered separately.

Software documentation is one of the biggest "hidden costs" in generating software systems. When it is produced in quantity, it rarely fulfills its functions, and when it is omitted as a cost-saving measure, the organization pays for the some software problem to be solved over and over. (21:80)

Figure 7 presents the computer software productivity data collected. Nelson's data indicates the expected trend of a higher documentation cost for a more complex scientific program. Because the large variations in the data are so large and little justification was provided by the sources no other trends can be identified.

Figure 8 shows the theoretical relationship of varying documentation costs to total project cost and hypothesizes an optimum documentation level.

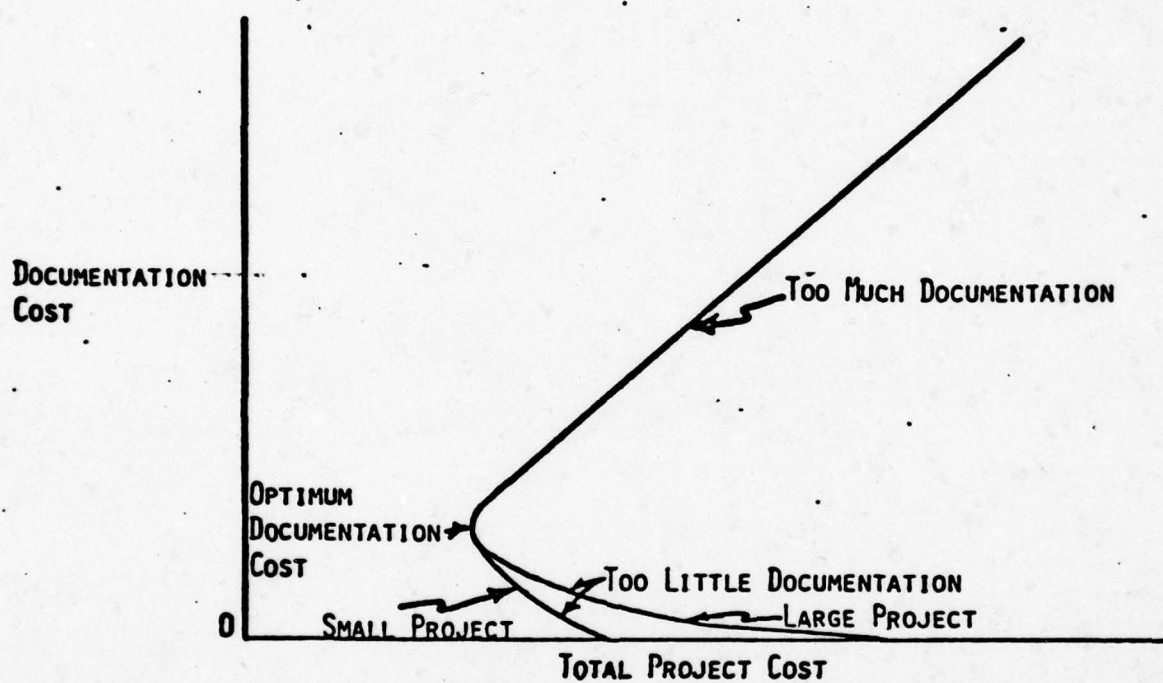
Computer Time

The computer time required is another factor that must be considered in the total software development cost. Figure 9 presents the data collected in this area. This data varies widely also. However, two expected trends are indicated. Greater computer time is required for complex programs using machine oriented language as opposed to simple programs using procedure orientened language.

Indirect costs can be incurred if an inefficient computer operation is used. For example, a programmer may have to wait around for hours or days to get a computer printout that he needs to move to the next step. Inefficiencies may also result if programmer man hours is traded off for computer time. That is where desk checking may be more efficient, the

	Instructions Per Page	Cost to Generate Per Page	Cost to Produce Per Page	Comments
Namus/Farr (14:243)	50			on 5 large scale prog.
Wood (18:154)	10-15	\$70-\$100	\$70-\$100	variation due to prog. size, type of prog., and standards of documentation
Morin (20:132)	28-100			
Nelson (3:68)	5.68 3.97 14.5			business, large program scientific, large program utility, large program
Company A	7.5	1 man day		IAW WS-8506
B	50			25% of total prog. effort
C				20% of total prog. effort
D				

Figure 7. Computer Software Documentation Productivity Information



RELATION OF DOCUMENTATION COST TO TOTAL PROJECT COST (18:155)

Figure 8.

	Computer time		Comments
	64 seconds/instruction		
Fleishman (19:15)	43	"	scientific application
	108	"	business application
	36	"	machine oriented language
		"	procedure oriented language
Wood (18:152)	40	"	
Wolverton (22:13)	72	"	large programs
Namus/Farr (14:243)	14	"	increases exponentially for large programs
Company A	1 hr/M coding thru checkout		batch
	1 hr/MW checkout		block

Figure 9. Computer Time for Programming

programmer keeps putting the program in the computer with only small changes without thinking the problem out. This can be particularly costly to the government if it is furnishing the computer time.

Task Breakdown

Figure 10 shows a breakdown of software development cost for tasks as a percent of the total programming effort. This figure was included to provide the reader with a feel for the relative effort normally required for testing of the program. This knowledge would seem to be particularly valuable when reviewing a contractors proposed development schedule. If he has allowed only 10% of this total effort for testing, we have a high probability for a schedule/cost growth.

Software Estimating Equations

Numerous equation using methods for estimating computer software development costs exist. The Krauss Method (24:100) is summarized and evaluated by Morin. (20:22)

Krauss describes a method for estimating programming time that he says he has found to be fairly accurate. The method is designed for use in application programming projects, particularly business programming projects, with no restrictions as to the size of the programs. Encompassed in the estimate of programming time is the time required for the programming tasks of designing, coding, testing, and cocumenting. The method is to be applied subsequent to the completion of the system design activity.

When applying this method, the estimator must use his judgment, experience, and knowledge to determine as accurately as possible the following five factors:

1. Size-the estimated number of computer instructions or the total storage requirements for absolute code.
2. Complexity-an indicator based upon logic and variations in processing.
3. Input/Output-the number and the kinds of devices to be controlled by the program.
4. Programming Language-the language to be used to code.
5. Programmer Know-How-an indicator of programmer experience.

	Analysis and Design	Coding	Testing	Documentation	Comments
Wood (18:149)	35%	20%	45%		
CCIP-85 (10:39)					
SAGE	39%	14%	47%		
NITS	30%	20%	50%		
GEMINI	36%	17%	47%		
SATURN V	32%	24%	44%		
Joslin-Kennevan (23:82)					
Delaney	25%	31%	31%	13%	Simple
RCA	30%	20%	45%	5%	Average
Brandon	11%	11%	50%	28%	Complex
	27.5%	24%	34.5%	14%	Autocoder
Test Group	30%	27%	33%	10%	Assembler
	11.9%	30%	39.9%	18.2%	
	7.3%	29.4%	46.9%	16.4%	
Company A	30%	40%	30%		
B	35%	20%	30%	+25%	
C	35%	15%	50%	+20%	
D	30%	20%	50%		
E	50%	30%	20%		
F	55%	15%	20%	10%	

Figure 10. Software Development Cost Breakdown, Tasks as a Percent of Total Effort.

Krauss constructed tables which can be used to determine values for the above five factors. See Figure 11 for a list of values for each factor. Krauss developed a formula which used these factors to estimate the number of man-days required to program. The formula is:

$$\text{Unadjusted Estimate (in man-days)} = \frac{(\text{Estimated Size Value} \times \text{Complexity Value}) + \text{Input/Output Values}}{\text{Programming Language Value}}$$

To account for individual differences, the following formula is applied:

$$\text{Adjusted Estimate} = \text{Programmer Know-How Allowance} \times \text{Unadjusted Estimate.}$$

Krauss added two caveats to the use of his formula.

1. If a program consists of a number of modules which are either written by different types of programmers, or vary in complexity, each module must be estimated separately.
2. Because his formula does not take into account the time that may be consumed in nonprogramming activities such as vacations, holidays, administrative duties, training, company meetings, presentations to management, etc., he suggests the use of an overall loss factor of 20 to 30 percent for such activities.

The Krauss Method's major flaw is its dependence on the estimator's ability to predict the size and complexity of the programs. Even though the value range for these two factors is relatively broad, it has been my experience that most estimators have difficulty predicting project size within 5,000 instructions of the actual program size before coding takes place. The values for programming size in Figure 11 also assume a linear relationship between effort. There is significant evidence that such linearity does not exist for programming systems. As a result of the assumption of linearity by Krauss, no size limitation was placed on the programs to which this method could be applied. However, I believe that the Krauss Method would be increasingly inaccurate as the programs increase in size.

On the surface Krauss's equation looks promising for use in the DOD Program Office, but let's look to see what happens when values that could be expected on a DOD minicomputer programming effort are assumed.

Figure 11.
Krauss's Values for Input Variables

<u>Estimated Size</u>	<u>Value Range</u>
1,000 - 5,000	1 - 5
6,000 - 10,000	6 - 10
11,000 - 15,000	11 - 15
-	-
-	-
-	-
96,000 - 100,000	96 - 100
-	-
-	-
-	-
<u>Complexity Rating</u>	<u>Value Range</u>
Low difficulty	1 - 2
Intermediate difficulty	3 - 6
Average difficulty	7 - 12
Above average difficulty	13 - 19
Very high difficulty	20 - 30
Experimental	31 - 50
<u>Kind of Input/Output Device</u>	<u>Value</u>
Card Reader	1
Card Punch	2
Printer	4
Console typewriter	4
Paper tape	6
Magnetic tape	8
Disk	10
Data cell	12
Drum	12
Optical or MICR reader	15
Typewriter terminal	15
Graphic terminal	15
Audio terminal	15
Film scanner	16
<u>Programming Language</u>	<u>Value</u>
Absolute	2
Assembler	5
COBOL	7
PL/I	7
RPG	9
FORTRAN	10
<u>Programmer Know-How</u>	<u>Allowance Range</u>
Senior Programmer	0.6 - 1.0
Programmer	0.9 - 1.4
Associate Programmer	1.2 - 1.6
Junior Programmer	1.4 - 1.8
Trainee Programmer	1.7 - 5.0

Assumed Values:

Estimated Program Size - 5,000 Instructions, Value = 5

Complexity Rating - Very High Difficulty, Value = 30

Kind of Input/Output Device - Console Typewriter/Printer, Value = 4

Programming Language - FORTRAN, Value = 10

Programmer Know How - Programmer, Value = 1

Loss Factor - 30%

Calculation:

Unadjusted Estimate (in man-days) =

$$\frac{(\text{Estimated Size Value} \times \text{Complexity Value}) + \text{Input/Output Values}}{\text{Programming Language Value}}$$

$$= \frac{5 \times 30 + 4 + 4}{10} = 15.8 \text{ man days}$$

Adjusted Estimate =

Programmer Know How Allowance x Unadjusted Estimate

$$= 1.0 \times 15.8 \text{ man days} = 15.8 \text{ man days}$$

With a loss factor, the estimate becomes 20.54 man days for 5000 instructions. This is approximately 243 instructions per man day, but does not include the analysis and design tasks efforts. Decreasing this by 40% (based on Figure 10) to compensate for analysis and design results in approximately 146 instructions per man day. The Krauss Method is designed for use on business programming projects. If a 50% reduction in efficiency is assumed for a scientific program the productivity rate decreases to 73 instructions per man day, which seems unrealistically high.

Because the most significant factors seem to be included in the Krauss equation I believe it could be used effectively by making adjustments to his input factors and adding factors for computer time and documentation.

Which Method?

This investigation leads me to the following course of action for determining the resources required for a computer software development effort.

1. A baseline equation should be established that considers programming effort, documentation effort and computer time requirements. The equation would be established by using the best possible data base (the data base that most closely compares to the new program).
2. Refine the baseline equation by collecting and incorporating historical data.

I don't believe the form of the equation is very important as long as it meets the criteria above. There are software estimating equations that are much more complex than Krauss's, but any equation will be useless to the Program Office if it is not based on accurate historical data from programs which are very similar to the one being estimated.

I propose the baseline equation below:

Program Development Cost in Dollars =
(Programming Effort) +
(Documentation Effort) +
(Computer Time) +
(Other Direct Costs) =

$$\frac{I}{R_p} ((S_p) + (S_p \times O_p)) +$$
$$\frac{I}{I_p} (((P_{ep}) ((S_p) + (S_p \times O_p))) + (P_p)) +$$
$$I (T_i) (C_c) +$$
$$(ODC) =$$

Where:

I = number of instructions in program
R_p = programmer production rate in instructions per hour
S_p = programmer hourly wage
O_p = programmer overhead rate
I_p = estimated number of instructions per page of documentation
P_{ep} = programmer effort per page of documentation (hours)
P_p = production cost per page of documentation (art work, typing, etc.)
T_i = computer time required (minutes per instruction)
C_c = Cost of computer time per minute

Initial values would be plugged into the equation based on the best information available, and then updated or scaled to different applications

based on an increasing data base.

If the Program Office has many different but similar programming efforts under contract it seems that a beneficial data base could be obtained rather quickly. This would seem especially true in the case of an office that utilizes many minicomputers. I hope that it has been sufficiently demonstrated earlier in this report how one variable can throw an estimate completely off, so great care must be taken to determine the differences between the data base programs and the new program.

Step 3 - Converting the Effort into Dollars

The discussion of actual dollar costs will be limited to presentation of some typical values as inputs to the proposed baseline equation.

$$\frac{I}{R_p} ((Sp) + (Sp \times Op)) +$$

$$\frac{I}{I_p} (((Pep) ((Sp) + (Sp \times Op))) + (Pp)) +$$

$$I (Ti) (Cc) +$$

$$(ODC) =$$

$$\frac{I}{I} ((\$10.00) + (\$10.00 \times 120\%)) +$$

$$\frac{I}{I_0} (((7) ((\$10.00) + (\$10.00 \times 120\%))) + (\$35.00)) +$$

$$(I \times .6 \times \$5.00) +$$

$$(ODC) =$$

$$(I \times \$22.00) + \left(\frac{I}{(10)} (\$154.00 + \$35.00) \right) + (I \times \$3.00) + (ODC) =$$

$$\$43.90 (I) + (ODC)$$

G & A / fee were not included in the equation above. Other Direct Costs cover such things as travel.

The values used in the example above are my best estimate of averages based on my limited research, and of course would have little validity for application for any specific situation. Each program must be considered individually.

VII. CONCLUSIONS AND RECOMMENDATIONS

The poor cost estimate results in a large percentage of the cost growth on DOD programs. The primary reason for poor estimates for computer software development is probably the lack of, and difficulty in establishing standards.

The cost of computer software development will be based on a number of factors, some of which are listed below.

1. Complexity of the program.
2. Efficiency of the programmers.
3. Size of the program.
4. Computer memory available to the programmer.
5. Level of documentation.
6. Quality/type of specification.
7. Type of language.

In order to make an accurate cost estimate for computer software development a good historical data base is required. Therefore the primary recommendation of this report is that the Program Office establish a data base which is broken down into as many cost contributing factors as possible, by:

1. Making computer software a separate line item in a software/hardware contract.
2. Require that the contractor provide a Work Breakdown Structure in accordance with Mil-Std-881 as part of his proposal, and then report costs against it.
3. Annotate the contractors report with information that will

better define the effort accomplished. For example, the skill level of the programmer.

Additional recommendations for the Program Office:

1. Contract for the Computer Program Performance Specification and Computer Program Design Specification separate from the actual programming effort.
2. Develop and update a baseline equation similar to the one described in section VI.

Recommendation for further study:

Conduct a thorough search to determine what computer software development data bases exist, and evaluate the success being obtained in using them in estimating.

BIBLIOGRAPHY

1. Weitzman, Gay; Minicomputer Systems, Structure, Implementation, and Application, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1974.
2. Burkhalter, K. E. Jr.; Characteristic Minicomputer Architecture, Minicomputers II: Hardware, Software and Systems, Professional Growth in Engineering Seminar, National Engineering Consortium, Inc., 1974.
3. Nelson, E. A.; Management Handbook for the Estimation of Computer Programming Costs, Systems Development Corporation, Santa Monica, California, March 20, 1967.
4. Melburn, Michael; Toward Full Disclosure of Program Status, The Federal Accountant, March 1974.
5. Logistics Management Institute; Introduction to Military Program Management; LMI Task 69-28, Washington, D.C., March 1971.
6. U.S. Air Force; Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980's (CCIP-85), Executive Summary, February 1972.
7. Air Force Systems Command; Project Ace - Findings and Action Plans - Progress Report, Andrews AFB, Nov. 1973.
8. Hanks, Dale; Programming Considerations for Minicomputers, Computers and People, January 1974.
9. Farquhar, J. A.; A Preliminary Inquiry Into the Software Estimating Process, RM-6271-PR, Rand, Santa Monica, California, August 1970.
10. U.S. Air Force; Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980's (CCIP-85), Vol. I, April 1972.
11. Goldberg, Jack, Editor; Proceedings of a Symposium on The High Cost of Software, Held at the Naval Postgraduate School, Monterey, California, Sept. 17-19 1973, Stanford Research Institute, Menlo Park, California.
12. Boehm, B.W.; Software and Its Impact: A Qualitative Assessment, Datamation, May 1973.
13. Keider, Stephen P.; Why Projects Fail, Datamation, December 1974.
14. Nanus, B., and Farr, L.; Some Cost Contributions to Large-Scale Programs, American Federation of Information Processing Societies, Inc., SJCC, 25, 1964.

15. Data Processing Manual, Auerbach Publishers, 1974.
16. Brooks, F. P.; The Mythical Man-Month, Datamation, December 1974.
17. U.S. Navy; Comparison and Summary of AN/UYK-7 Combat System Testing (on line and off line) Rev A., Naval Ship Engineering Center, Hyattsville, Md., July 1974.
18. Wood, D. L.; Data Processing Value Engineering, Society of American Value Engineering Proceedings, Meeting of May 13-16, 1973.
19. Fleishman, T.; Current Results From the Analysis of Cost Data for Computer Programming, Electronic Systems Division, Air Force Systems Command, L.G. Hanscom Field Bedford, Mass., August 1966.
20. Morin, Lois H.; Estimation of Resources for Computer Programming Projects, M.S. Thesis, University of North Carolina, Chapel Hill, 1974.
21. Ridge, Warren J. and Johnson, Leann E.; Effective Management of Computer Software, Dow Jones-Irwin, Inc., Homewood, Illinois, 1973.
22. Wolverton, Ray W.; The Cost of Developing Large Scale Software, Paper prepared for IEEE 1972 International Convention and Exposition, New York, March 20-23, 1972.
23. Kennevan, W. J. and Joslin, E. O.; Management and Computer Systems, College Readings Inc., 1973.
24. Krauss, Leonard; Administering and Controlling the Company Data Processing Function, Prentice-Hall, Inc., Englewood, N.J. 1969.

GLOSSARY OF DIGITAL COMPUTER TERMS (2:A-8)

Absolute - Pertaining to an address fully defined by a memory address number, or to a program which contains such addresses (as opposed to one containing symbolic addresses).

Accumulator - A register in which numbers are totaled, manipulated, or temporarily stored for transfers to and from memory or external devices.

Add - Restrictive: "two's complement" addition of binary numbers.
General: any arithmetic addition.

Address - (Noun) A number which identifies one location in memory.
(Verb) To direct the computer to read a specified memory location (synonymous with "reference").

Address Modification - A programming technique of changing the address specified by a memor-reference instruction, so that each time that particular instruction is executed, it will affect a different memory location.

Address Word - A computer word which contains only the address of a memory location.

ALGOL - Algebraic-Oriented Language - An international algebraic procedural language for a computer programming system.

Algorithm - A prescribed set of well-defined rules or processes for the solution of a problem in a finite number of steps.

Alphanumeric - Pertaining to a character set that contains both letters and numerals, and usually other characters.

Alter - A modification of the contents of an accumulator or extend bit, e.g., clear, complement, or increment.

"Add" - A logical operation in which the resultant quantity (or signal) is true if all of the input values are true, and is false if at least one of the input values is false.

Argument - 1) A variable or constant which is given in the call of a subroutine as information to it. 2) A variable upon whose value the value of a function depends. 3) The known reference factor necessary to find an item in a table or array i.e., the index.

Arithmetic Logic - The circuitry involved in manipulating the information contained in a computer's accumulators.

Arithmetic Operation - Restrictive: A mathematical operation involving fundamental arithmetic (addition, subtraction, multiplication, division), specifically excluding logical and shifting operations.
General: any manipulation of numbers.

Array - A set of lists of elements, usually variables or data.

ASCII - An abbreviation for American Standard Code for Information Interchange.

Assemble - To translate from a symbolic program to a binary program by substituting binary operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

Assembler - A program for a computer which converts a program prepared in symbolic form (i.e., using defined symbols and mnemonics to represent instructions, addresses, etc.) to binary machine language.

Assembly Language - The source language used as input to an assembler and translated by the assembler into machine language.

Auxiliary Storage - Storage that supplements core memory, such as disk tape.

Background Processing - The automatic execution of a low priority computer program when higher priority programs are not using the system resources.

Base - The quantity of different digits used in a particular numbering system. The base in the binary numbering system is two; thus there are two digits (0 and 1). In the decimal system (base 10), there are ten digits (0 through 9).

Base Address - A given address from which an absolute address is derived by combination with a relative address.

Base Page - The lowest numbered page of a computer's memory. It can be directly addressed from any other page.

Binary - Denoting the numbering system based on the radix two. Binary digits are restricted to the values 0 and 1.

Binary Coded Decimal (BCD) - A coding method for representing each decimal digit (0-9) by specific combinations of four bits. For example, the 8-4-2-1 bed code commonly used with computers represents "1" as 0001, and "9" as 1001.

Binary Program - A program (or its recording form) in which all information is in binary machine language.

Bistable - Pertaining to an electronic circuit having two stable states, controllable by external switching signals, analogous to an on-off switch.

Bit (b) - A single digit in a binary number, or in the recorded representation of such a number (by hole punches, magnetic states, etc). The digit can have one of only two values, 0 or 1.

Bit Density - A physical specification referring to the number of bits which can be recorded per unit of length or area.

Bit serial - One bit at a time, as opposed to bit parallel in which all bits of a character can be handled simultaneously.

Block - A set of consecutive machine words, characters, or digits handled as a unit, particularly with reference to I/O.

Bootstrap - A technique or device designed to bring itself into a desired state by means of its own action, e.g., a routine whose first few instructions are sufficient to bring the rest of itself into the computer from an input device.

Branch - A point in a routine where one of two or more choices is made under control of the routine.

Breakpoint - A point in a computer program at which conditional interruption is made to permit visual check, printouts, or other debugging aids.

Buffer - A register used for intermediate storage of information in the transfer sequence between the computer's accumulators and a peripheral device or a designated area of memory used to temporarily hold data.

Bug - A mistake in the design or implementation of a program resulting in erroneous results.

Bulk Memory - Storage in addition to the main memory of the computer, e.g., magnetic tape, disc or drum.

Bus - A major electrical path connecting two or more electrical circuits.

Byte - A group of binary digits usually operated upon as a unit, frequently eight b.

Calling Sequence - A specified set of instructions and data necessary to set up and call a given routine.

Carry - A digit, or equivalent signal, resulting from an arithmetic operation which causes a positional digit to equal or exceed the base of the effective numbering system.

Central Processing Unit (CPU) - The unit of a computing system that includes the circuits controlling the interpretation and execution of instructions--the computer proper, excluding I/O and other peripheral devices.

Character - The general term to include all symbols such as alphabetic letters, numerals, punctuation marks, mathematical operators, etc. Also, the coded representation of such symbols.

Checkpoint - A point in time during a program run at which processing is momentarily halted to make a record, on an external storage medium of the condition of the variables of the program being executed.

Clear - To erase the contents of a storage location by replacing the contents, normally with zeros or spaces; to set to zero.

Code - A system of symbols which can be used by machines, such as a computer, and which in specific arrangements have a special external meaning.

Coding - Writing instructions for a computer using symbols meaningful to the computer, or to an assembler, compiler, or other language processor.

Compatability - The ability of an instruction or source language to be used on more than one computer.

Compile - To produce a binary-coded program from a program written in source (symbolic) language, by selecting appropriate subroutines from a subroutine library, as directed by the instructions or other symbols of the source program. Linkage information is supplied for combining the subroutines into a workable program, and the subroutines and linkage are translated into binary code.

Compiler - A language translation program, used to transform symbols meaningful to a human operator to codes meaningful to a computer. More restrictively, a program which translates a machine-independent source language into the machine language of a specific computer, thus excluding assemblers.

Complement - (One's) To replace all bits with 1 bits and vice versa.
(Two's) To form the one's complement and add 1.

Computation - The processing of information within the computer.

Computer (digital) - An electronic instrument capable of accepting, storing, and arithmetically manipulating information, which includes both data and the controlling program. The information is handled in the form of coded binary digits (0 and 1), represented by dual voltage levels, magnetic states, punched holes, etc.

Computer Word - See "word".

Conditioned Assembly - Assembly of certain parts of a symbolic program only if certain conditions have been met.

Configuration - The arrangement of either hardware instruments or software routines when combined to operate as a system.

Console - Usually the external front side of a device, where controls and indicators are available for manual operation of the device.

Constant - Numeric data used but not changed by the program.

Contents - The information stored in a register or memory location.

Convert - 1) To change numeric data from one radix to another. 2) To transfer data from one recorded format to another.

Core - The smallest element of a core storage memory module. It is a ring of ferrite material that can be magnetized in clockwise or counterclockwise directions to represent the two binary digits, 0 and 1.

Core Memory - The main high-speed storage of a computer, in which binary data is represented by the switching polarity of magnetic cores.

Current Location Counter - A counter kept by an assembler to determine the address assigned to an instruction or constant being assembled.

Current Page - The memory page comprising all those locations which are on the same page as a given instruction.

Cycle Time - The length of time it takes the computer to reference one word of memory.

Data - A general term used to denote any or all facts, numbers, letters, and symbols. It connotes basic elements of information which can be processed or produced by a computer.

Data Acquisition - The gathering, measuring, digitizing, and recording of continuous-form (analog) information.

Data Reduction - The transformation of raw information gathered by measuring or recording equipment into a more condensed, organized, or useful form.

Data Word - A computer word consisting of a number, a fact, or other information which is to be processed by the computer.

Debug - To check for and correct errors in a program.

Decimal - Denoting the numbering system based on the radix ten.

Decrement - To change the value of a number in the negative direction.
If not otherwise stated, a decrement by one is usually assumed.

Device - An electronic or electromechanical instrument. Most commonly implies measuring, reading, or recording equipment.

Diagnostic - (Adjective) Relating to test programs for detection of errors in the functioning of hardware or software, or the messages resulting from such tests. (Noun) The test program or message itself.

Digit - A character used to represent one of the non-negative integers smaller than the radix, e.g., in binary notation, either 0 or 1.

Direct Address - An address that specifies the location of an instruction operand.

Direct Memory Access - A means of transferring a block of information words directly between an external device and the computer's memory, bypassing the need for repeating a service routine for each word. This method greatly speeds the transfer process.

Disable - A signal condition which prohibits some specific event from proceeding.

Disc Storage - A means of storing binary digits in the form of magnetized spots on a circular metal plate coated with a magnetic material. The information is stored and retrieved by read-write heads which may be positioned over the surface of the disc either by moving the heads or the disc itself.

Documentation - Manuals and other printed materials (tables, listings, diagrams, etc.) which provide instructive information for usage and maintenance of a manufactured product, including both hardware and software.

Double-length Word - A word which, due to its length, requires two computer words to represent it. Double-length words are normally stored in two adjacent memory locations.

Double Precision - Pertaining to the use of two computer words to represent one number.

Downtime - The time interval during which the device is inoperative.

Dummy - Used as an adjective to indicate an artificial address, instruction, or record of information inserted solely to fulfill prescribed conditions, as in a "dummy" variable.

Dump - To copy the contents of all or part of core memory, usually onto an external storage medium.

Dynamic Relocation - The ability to move programs or data from auxiliary memory into main memory at any convenient location. Normally the addresses of programs and data are assigned when the program is compiled.

Effective Address - The address of a memory location ultimately affected by a memory reference instruction. It is possible for one instruction to go through several indirect addresses to reach the effective address.

Enable - A signal condition which permits some specific event to proceed, whenever it is ready to do so.

"Exclusive-Or" - A logical operation in which the resultant quantity (or signal) is true if at least one (but not all) of the input values is true, and is false if the input values are all true or all false.

Execute - To fully perform a specific operation, such as would be accomplished by an instruction or a program.

Exit Sequence - A series of instructions to conclude operation in one area of a program and to move to another area.

External Storage - A separate facility or device on which data usable by the computer are stored (such as paper tape, tape, or disk).

Field - 1) One or more characters treated as a unit. 2) A specified area of a record used for a single type of data.

File - A collection of related records treated as a unit.

Filename - Alphanumeric characters used to identify a particular file.

Fixed point - A numerical notation in which the fractional point (whether decimal, octal, or binary) appears at a constant predetermined position. Compare with "floating point."

Flag - A variable or register used to record the status of a program or device - in the latter case sometimes called a "device flag."

Flip-Flop - An electronic circuit having two stable states, and thus capable of storing a binary digit. Its states are controlled by signal levels at the circuit input and are sensed by signal levels at the circuit output.

Floating Point - A numerical notation in which the integer and the exponent of a number are separately represented (frequently by two computer words), so that the implied position of the fractional point freely varied with respect to the integer digits. Compare with "fixed point."

Flowchart - A diagram representing the operation of a computer program.

Foreground Processing - Higher priority processing that takes precedence over "background processing" and can interrupt such processing. It results from real-time events or enquiries.

Format - A predetermined arrangement of bits and characters.

FORTRAN - A programming language (or the compiler which translates this language) which permits programs to be written in a form resembling algebra, rather than in detailed instruction by instruction format.

Forward Referencing - The need to refer to a symbol in a program prior to its definition (i.e., trying to assemble the instruction JUMP PLACE, where PLACE is a location symbol further down in the program code).

Full-Duplex - Describing a communicational channel capable of simultaneous and independent transmission and reception.

Gate - An electronic circuit capable of performing logical functions such as "and", "or", "nor", etc.

Half-Duplex - Describing a communication channel capable of transmission and/or reception, but not both simultaneously.

Hardware - Electronic or electromechanical components, instruments or, systems.

High Core - Core-memory locations having high-numbered addresses.

"Inclusive-Or" - A logical operation in which the resultant quantity (or signal) is true if at least one of the input values is true, and is false if the input values are all false.

Increment - To change the value of a number in the positive direction. If not otherwise stated, an increment by one is usually assumed.

Incremental Magnetic Tape - A form of magnetic tpe recording in which the recording transport advances by small increments (e.g., 0.005 in.), stopping the tape advancement long enough to record one character at the spot located under the recording head.

Index Register - A memory device containing an index. See "Address Modification."

Indirect Address - The address initially specified by an instruction when it is desired to use that location to re-direct the computer to some other location to find the "effective address" for the instruction.

Information - A unit or set of knowledge represented in the form of discrete "words," consisting of an arrangement of symbols or (so far the digital computer is concerned) binary digits.

Inhibit - To prevent a specific event from occurring.

Initialize - The procedure for setting various parts of a stored program to starting values, so that the program will behave the same way each time it is repeated. The procedures are included as part of the program itself.

Input - Information transferred from a peripheral device into the computer. Also applied in the transfer process itself.

Input/Output (I/O) - Relating to the equipment or method used for transmitting information into and out of the computer.

Input/Output Channel - The complete input or output facility for one individual device or function, including its assigned position in the computer, the interface circuitry, and the external device.

Instruction - A witten statement or the equivalent computer-acceptance code, which tells the computer to execute a specified single operation.

Instruction Code - The arrangement of binary digits which tell the computer to execute a particular instruction.

Instruction Logic - The circuitry involved in moving binary information between registers, memory, and buffers in prescribed manners, according to instruction codes.

Instruction Word - A computer word containing an instruction code. The code bits may occupy all or (as in the case of memory reference instruction words) only part of the word.

Interface - The connecting circuitry which links the central processor of a computer system to its peripheral devices.

Internal Storage - The storage facilities forming an integral physical part of the computer and directly controlled by the computer. Also called "main memory" and "core memory."

Interpreter - A program which translates and executes source language statements at run time.

Interrupt - The process, initiated by an external device, which causes the computer to interrupt a program in progress, generally for the purpose of transferring information between that device and the computer.

Interrupt Location - A memory location whose contents (always an instruction) are executed upon interrupt by a specific device.

Iteration - Repetition of a group of instructions.

Job - A unit of code which solves a problem, i.e., a program and all its related subroutines and data.

Jump - An instruction which breaks the strict sequential location-by-location operation of a program and directs the computer to continue at another specified location anywhere in memory.

K - One thousand twenty four. For example, 4k words of memory means 4096 words.

Label - Any arrangement of symbols, usually alphanumeric, used in place of an absolute memory address in computer programming.

Language - The set of symbols, rules, and conventions used to convey information, either at the human level or at the computer level.

Leader - The blank section of tape at the beginning of the tape.

Least Significant Digit - The rightmost digit of a number.

Library Routine - A routine designed to accomplish some commonly used mathematical function and kept permanently available on a library program tape (e.g., FORTRAN Library).

Line Feed - A terminal or line printer operation which advances the the paper by one line.

Line Number - In source languages such as BASIC and FORTRAN, a number which begins a line of the source program for purposes of identification. A numeric label.

Linkage - In programming, code that connects two separately coded routines.

List - 1) A set of items. 2) To print out a listing on the line printer or terminal. 3) See "Pushdown list."

Literal - A symbol which defines itself.

Load - To put information into (memory, a register, etc.). Also (e.g., loading tape), to put information medium into the appropriate device.

Loader - A program designed to assist in transferring information from an external device into a computer's memory.

Load Time - That time at which an assembled program is placed in the computer and readied for execution.

Location - A group of storage elements in the computer's memory which can store one computer word. Each such location is identified by a number ("address") to facilitate storage and retrieval of information in selectable locations.

Logical Operation - A mathematical process based on the principles of truth tables, e.g., "and", "inclusive-or", and "exclusive-or" operations.

Logic Diagram - A diagram which represents the detailed internal functioning of electronic hardware, using binary logic symbols rather than electronic component symbols.

Logic Equation - A written mathematical statement, using symbols and rules derived from Boolean algebra. Specifically (hardware design), a means of stating the conditions required to obtain a given signal.

Loop - A sequence of instructions in which the last instruction is a jump back to the first instruction.

Low Core - Core-memory locations having low-numbered addresses.

Machine - Pertaining to the computer hardware (e.g., machine timing, machine language).

Machine Language - The form of code information (consisting of binary digits) which can be directly accepted and used by the computer. Other languages require translation to this form, generally with the aid of translation programs (assemblers and compilers).

Machine Timing - The regular cycle of events in the operation of internal computer circuitry. The actual events will differ for various processes, but the timing is constant through each recurring cycle.

Macro - An assembly-time facility that allows lines of text to be retrieved and modified by the substitution of text for dummy names in the saved text. The resulting modified text is assembled at the point of retrieval.

Macroinstruction - An instruction, similar in binary coding to the computer's basic machine-language instructions, which is capable of producing a variable number of machine-language instructions.

Magnitude - That portion of a computer word which indicates the absolute value of a number, thus excluding the sign bit.

Mask - A bit pattern which selects those bits from a word of data which are to be used in some subsequent operation.

Mass Storage - Pertains to a device, such as tape or disk, which stores large amounts of data readily accessible to the central processing unit.

Media Conversion - The transferral of recorded information from one recording medium (e.g., punched paper tape, magnetic tape, etc.) to another recording medium.

Memory - An organized collection of storage elements (e.g., ferrits cores), into which a unit of information consisting of a binary digit can be stored, and from which it can later be retrieved. Also, a device not necessarily having individual storage elements, but which has the same storage and retrieval capabilities (e.g., magnetic discs).

Memory Cycle - That portion of the computer's internal timing during which the contents of one location of memory are read out (into the Transfer Register) and written back into that location.

Memory Module - A complete segment of core storage, capable of storing a definable number of computer words (e.g., 4096 or 8192 words). Computer storage capacity is incremental by modules and is frequently rounded off and abbreviated as "4k" (e.g., 4096 or approximately 4000 words), "8k" (8192 or 8000), "16k", etc.

Memory Protect - A means of preventing inadvertent alteration of a selectable segment of memory.

Memory Reference - The address of the memory location specified by a memory-reference instruction, i.e., the location affected by the instruction.

Microcomputer - A general term used to describe computers or major parts of a computer when they are implemented on LSI chips.

Microinstruction - An instruction which forms part of a larger composite instruction.

Minicomputer - A general term used to describe small computers. In this sense, small usually implies both the computer's physical size and its word size (data-path width). Most minicomputers are designed with a 16 bit word size, but sizes from 8 to 19 bits are considered in the minicomputer range.

Monitor - An operating programming system which provides a uniform method for handling the real-time aspects of program timing, such as scheduling and basic input/output functions.

Most Significant Digit - The leftmost nonzero digit.

Multi-Level Indirect - Indirect addressing using two or more indirect addresses in sequence to find the effective address for the current instruction.

Multiple-Precision - Referring to arithmetic in which the computer, for greatest accuracy, uses two or more words to represent one number.

Multiprocessing - Utilization of several computers or processors to logically or functionally divide jobs or processes, and to execute them simultaneously.

Multiprogramming - A system of execution of two or more programs kept in core at the same time. Execution cycles between the programs.

Normalize - To adjust the exponent and fraction of a floating-point quantity so that the fraction appears in a prescribed format.

Object Programming - The binary coded program which is the output after translation from the source language; the binary program which runs on the computer.

Octal - Denoting a numbering system based on the radix eight. Octal digits are restricted to the values 0 through 7.

Octal Code - A notation for writing machine-language programs with the use of octal numbers instead of binary numbers.

Off-Line - Pertaining to the operation of peripheral equipment not under control of the computer.

One's Complement - A number so modified that the addition to the modified number and its original value, plus one, will equal an even power of two. A one's complement number is obtained mathematically by subtracting the original value from a string of 1's, and electronically by inverting the states of all bits in the number.

On-Line - Pertaining to the operation of peripheral equipment under computer control.

Operand - That which is effected, manipulated, or operated upon. The address or symbolic name, portion of an assembler instruction.

Operating System - An integrated collection of routines for supervising the sequencing of programs by a computer, e.g., debugging, input/output, operation, compilation, and storage assignment.

Operation (OP) Code - That part of an instruction designating the operation to be performed.

Operator - That symbol or code which indicates an action (or operation) to be performed.

Optisum Code - A set of machine language instructions which is particularly efficient with regard to a particular aspect, e.g., minimum time to execute or minimum or efficient use of storage space.

"Or" - (Inclusive) A logical operation such that the result is true if either or both operands are true, and false if both operands are false. Exclusive) A logical operation such that the result is true if either operand is true, and false if both operands are either true or false.

Origin - The absolute address of the beginning of a section of code.

Output - Information transferred from the computer to a peripheral device. Also applied to the transfer process itself.

Overflow - A condition that occurs when a mathematical operation yields a result whose magnitude is larger than the program is capable of handling.

Overlay - The operation of bringing into main memory and executing a segment which is a subprogram (i.e., a more or less separate entity) of a larger program.

Packed Word - A computer word containing two or more independent units of information. This is done to conserve storage when information requires relatively few bits of the computer word.

Page - An artificial division of memory consisting of a fixed number of locations, dictated by the direct addressing range of memory reference instructions.

Page Zero - The memory page which includes the lowest numbered memory addresses.

Parity Bit - A supplementary bit added to an information word to make the total of one-bits always odd or even. This permits checking the accuracy of information transfers.

Pass - The complete process of reading a set of recorded information (one tape, one set of cards, etc.) through an input device, from beginning to end.

Patch - To modify a routine in a rough or expedient way.

Peripheral Device - An instrument or machine electrically connected to the computer, but which is not part of the computer itself.

Plane - An arrangement of ferrite cores on a matrix of control and sensing wires. Several planes stacked together form a "memory module."

Pointer Address - Address of a core-memory location containing the actual (effective) address of desired data.

Power Failure Control - A means of sensing primary power failure so that a special routine may be executed in the finite period of time available before the regulated dc supplies discharge to unusable levels. The special routine may be used to preserve the state of a program in progress, or to shut down external processes.

Priority - The automatic regulation of events so that chosen actions will take precedence over others in cases of timing conflict.

Procedure - The course of action taken for the solution of a problem; also called an "algorithm."

Process Control - Automatic control of manufacturing processes by use of a computer.

Processor - The central unit of a computer system (i.e., the device which accomplishes the arithmetic manipulations), exclusive of peripheral devices. Frequently (when used as an adjective) also excludes interface components, even though normally contained within the processor unit; thus "processor" options exclude interface ("input/output") options.

Program - A plan for the solution of a problem by a computer, consisting of a sequence of computer instructions.

Program Listing - A printed record (or equivalent binary-output program) of the instructions in a program.

Programmer - A person who writes computer programs. Also (hardware), an interface card or instrument which sets up (or "programs") the various functions of one measuring instrument.

Programming - The process of creating a program.

Pseudo Instruction - A symbolic statement, similar to assembly-language instructions in general form, but meaningful only to the program containing it, rather than to the computer as a machine instruction.

Punched Tape - A strip of tape, usually paper, on which information is represented by coded patterns of holes punches in columns across the width of the tape. There are commonly 8 hole positions (channels) across the tape.

Pushdown List - A list that is constructed and maintained so that the next item to be retrieved is the item most recently started in the list.

Queue - A waiting list. In timesharing, the monitor maintains a queue of user programs waiting for processing time.

Radix - The base of a number system, the number of digit symbols required by a number system. See "binary," "octal."

Random-Access - Pertaining to a storage device in which the accessibility of data is effectively independent of the location of the data. (Synonymous with "direct-access").

Read - The process of transferring information from an input device into the computer. Also, the process of taking information out of the computer's memory. (see "memory cycle").

Real Time - Time elapsed between events occurring externally to the computer. A computer which accepts and processes information from one such event and is ready for new information before the next event occurs is said to operate in a "real-time environment."

Record - A collection of related items of data, treated as a unit.

Recursive Subroutine - A subroutine capable of calling itself and returning at some later point to the program which initially called it.

Reentrant Code - A program segment (e.g., subroutine) which can be executed (i.e., reentered) by more than one other program simultaneously. This mode of operation requires a separate storage area for storing information that varies for each instance of execution.

Register - An array of hardware binary circuits (flip-flops, switches, etc.) for temporary storage of information. Unlike mass storage of devices such as memory cores, registers can be wired to permit flexible control of the contained information, for arithmetic operations, shifts, transfers, etc.

Relative Address - The number that specifies the difference between the actual address and a base address.

Relocatable - Pertaining to programs whose instructions can be loaded into any stated area of memory.

Relocating Loader - A computer program capable of loading and combining relocatable programs (i.e., programs having symbolic rather than absolute addresses).

Reset - A signal condition representing a binary "zero."

Response Time - Time between initiating some operation from a terminal and obtaining results. Includes transmission time to the computer, processing time, access time to file records needed, and transmission time back to the terminal.

Restart - To resume the execution of a program.

Rotate - A positional shift of all bits in an accumulator (and possibly an extend bit as well) with those bits lost off one end of the accumulator "rotated" around to enter vacated positions at the other end.

Routine - A program or program segment designed to accomplish a single function.

Run Time - The time during which a program is executed.

Segment - 1) That part of a long program which may be resident in core at any one time. 2) To divide a program as in 1, or into two or more segments, or to store part of a program or routine on an external storage device to be brought into core as needed.

Serial-Access - Pertaining to the sequential or consecutive transmission of data to or from core, for example, paper tape. Contrast with "random-access."

Service Routine - A sequence of instructions designed to accomplish the transfer of information between a particular device and the computer.

Set - A signal condition representing a binary "one."

Shift - Restrictive (arithmetic shift): to multiply or divide the magnitude portion of a word by a power of two, using a positional shift of these bits. General: any positional shift of bits.

Sign - The algebraic plus or minus indicator for a mathematical quantity. Also, the binary digit or electrical polarity representing such an indicator.

Significant Digit - A digit so positioned in a numeral as to contribute a definable degree of precision to the numeral. In conventional written form, the most significant digit in a numeral is the leftmost digit, and the least significant digit is the rightmost digit.

Simulate - To represent the functioning of a device, system, or computer program with another system or program.

Skip - An instruction which causes the computer to omit the instruction in the immediately following location. A skip is usually arranged to occur only if certain specified conditions are sensed and found to be true, thus allowing various decisions be made.

Snapshot Dump - A dynamic printout during execution, at breakpoints and checkpoints, of selected areas in storage.

Software - Computer programs. Also, the tapes or cards on which the programs are recorded.

Software Package - A complete collection of related programs, not necessarily combined as a single entity.

Source Program - A program (or its recorded form) written in some programming language other than machine language and thus requiring translation. The translated form is the "object program."

Starting Address - The address of a memory location in which is stored the first instruction on a given program.

Statement - An instruction in any computer-related language other than machine language.

Storage Allocation - The assignment of blocks of data and instructions to specified blocks of storage.

Storage Capacity - The amount of data that can be entered, retained, and retrieved.

Storage Device - A device in which data can be entered, retained, and retrieved.

Store - To put information into a memory location, register, or device capable of retaining the information for later access.

String - A connected sequence of entities, such as characters in a command string.

Subroutine - A sequence of instructions designed to perform a single task, with provisions included to allow some other program to cause execution of the task sequence as if it were part of its own program.

Subscript - A value used to specify a particular item in an array.

Swapping - In a timesharing environment, the action of either temporarily bringing a user program into core or storing it on the disk or other system device.

Switch - A device or programming technique for making selections.

Symbol Table - A table in which symbols and their corresponding values are recorded.

Symbolic Address - A label assigned in place of absolute numeric addresses, usually for purposes of relocation. (See "relocatable.")

Symbolic Coding - Broadly, any coding or programming system in which symbols other than actual machine operations and addresses are used.

Symbolic Instructions - An instruction which is the basic component of an assembly language (input to assembler) and is directly translatable into machine language.

Syntax - 1) The structure of expressions in a programming language. 2) The rules governing the structure of a programming language.

Table - A collection of data stored for ease of reference, generally an array.

Temporary Storage - Storage locations reserved for immediate results.

Terminal - A peripheral device in a system through which data can either enter or leave the computer.

Timesharing - A method of allocating central-processor time and other computer services to multiple users so that the computer, in effect, processes a number of programs simultaneously.

Time Slicing - A method of job scheduling in a multiprogrammed system. This refers to the allocation of fixed amounts of computing time among users on a round-robin basis. Interrupts are generated by a fixed interval timer causing control to pass to the next waiting service request.

Toggle - Using switches to enter data into the computer memory.

Transfer Vector - A table, usually at a fixed location in memory, containing jump instructions and/or indirect addresses for jump instruction. When a jump instruction to a particular routine or when the address of a routine is placed in this table, other routines can call the routine without necessarily knowing its actual location in memory. This technique is used frequently when a relocatable assembler is not available for a particular machine.

Truncation - The reduction of precision by dropping one or more of the least significant digits; e.g., 3.141592 truncated to 4 decimal digits is 3.141.

Truth Table - A table listing of all possible configurations and resultant values for any given Boolean algebra function.

Two's Complement - A number so modified that the addition of the modified number and its original value will equal an even power of two. Also, a kind of arithmetic which represents negative numbers in two's complement form so that all addition can be accomplished in only one direction (positive incrementation). A two's complement number is obtained mathematically by subtracting the original value from an appropriate power of the base two, and electronically by inverting the states of all bits in the number and adding one (complement and increment).

Underflow - A condition that occurs when a floating-point operation yields a result whose magnitude is smaller than the program is capable of handling.

Updated Program - A program to which additions, deletions, or corrections have been made.

User - The person or persons who program and operate a particular computer.

Utility Routine - A standard routine to assist in the operation of the computer (e.g., device drivers, sorting routines, etc.) as opposed to mathematical ("library") routines.

Variable - A symbol whose value changes during execution of a program.

Waiting Loop - A sequence of instructions (frequently only two) which are repeated indefinitely until a desired external event occurs, such as the receipt of a Flag signal.

Word - A set of binary digits handled by the computer as a unit of information. Its length is determined by hardware design, e.g., the number of cores per location, and the number of flip-flops per register.

• Word Length - The number of bits in a word.

• Working Register - A register whose contents can be modified under control of a program. This a register consisting of manually operated switches is not considered a working register.

Write - The process of transferring information from the computer to an output device. Also, the process of storing (or restoring) information into the computer's memory (see "memory cycle").